协程asyncio_深入理解asyncio(二)

原创 weixin_39791653 于 2020-12-24 04:21:12 发布 63 收藏 版权

文章标签: 协程asyncio

Asyncio.gather vs asyncio.wait

在上篇文章已经看到多次用 asyncio.gather了，还有另外一个用法是 asyncio.wait，他们都可以让多个 协程 并发执行。那为什么提供2个方法呢？他们有什么区别，适用场景是怎么样的呢？其实我之前也是有点困惑，直到我读了asyncio的源码。我们先看2个协程的例子:

```
async def a():
print('Suspending a')
await asyncio.sleep(3)
print('Resuming a')
return 'A'
async def b():
print('Suspending b')
await asyncio.sleep(1)
print('Resuming b')
return 'B'
```

在IPython里面用gather执行一下:

```
In : return_value_a, return_value_b = await asyncio.gather(a(),
b())

Suspending a

Suspending b

Resuming b

Resuming a

In : return_value_a, return_value_b

Out: ('A', 'B')
```

Ok， asyncio.gather方法的名字说明了它的用途，gather的意思是「搜集」，也就是能够收集协程的结果，而且要注意，它会按输入协程的顺序保存的对应协程的执行结果。

接着我们说 asyncio.await，先执行一下：

```
In : done, pending = await asyncio.wait([a(), b()])

Suspending b

Suspending a

Resuming b

Resuming a

In : done

Out:

{<Task finished coro=<a() done, defined at <ipython-input-5-
5ee142734d16>:1> result='A'>,

<Task finished coro=<b() done, defined at <ipython-input-5-
5ee142734d16>:8
```

```
In : pending

Out: set()

In : task = list(done)[0]

In : task

Out: <Task finished coro=<b() done, defined at <ipython-input-5-
5ee142734d16>:8> result='B'>

In : task.result()

Out: 'B'
```

asyncio.wait的返回值有2项，第一项表示完成的任务列表(done)，第二项表示等待(Future)完成的任务列表(pending)，每个任务都是一个Task实例，由于这2个任务都已经完成，所以可以执行 task.result()获得协程返回值。

Ok, 说到这里，我总结下它俩的区别的第一层区别:

1. asyncio.gather封装的Task全程黑盒，只告诉你协程结果。
2. asyncio.wait会返回封装的Task(包含已完成和挂起的任务)，如果你关注协程执行结果你需要从对应Task实例里面用result方法自己拿。

为什么说「第一层区别」， asyncio.wait看名字可以理解为「等待」，所以返回值的第二项是pending列表，但是看上面的例子，pending是空集合，那么在什么情况下，pending里面不为空呢？这就是第二层区别：
asyncio.wait支持选择返回的时机。

asyncio.wait支持一个接收参数 return_when，在默认情况下，asyncio.wait会等待全部任务完成(returnwhen='ALLCOMPLETED')，它还支持FIRSTCOMPLETED(第一个协程完成就返回)和FIRSTEXCEPTION(出现第一个异常就返回):

```
In : done, pend
return_when=asy
```

```
Suspending a

Suspending b

Resuming b

In : done

Out: {<Task finished coro=<b() done, defined at <ipython-input-5-
5ee142734d16>:8> result='B'>}

In : pending

Out: {<Task pending coro=<a() running at <ipython-input-5-
5ee142734d16>:3> wait_for=<Future pending cb=
[<TaskWakeupMethWrapper object at 0x108065e58>()]>>}
```

看到了吧，这次只有协程b完成了，协程a还是pending状态。

在大部分情况下，用asyncio.gather是足够的，如果你有特殊需求，可以选择asyncio.wait，举2个例子：

1. 需要拿到封装好的Task，以便取消或者添加成功回调等
2. 业务上需要FIRSTCOMPLETED/FIRSTEXCEPTION即返回的

asyncio.createtask vs loop.createtask vs asyncio.ensure_future

创建一个Task一共有3种方法，如这小节的标题。在上篇文章我说过，从Python 3.7开始可以统一的使用更高阶的 `asyncio.create_task`。其实 `asyncio.create_`

```
def create_task(coro):

loop = events.get_running_loop()

return loop.create_task(coro)
```

loop.create_task接受的参数需要是一个协程，但是asyncio.ensure_future除了接受协程，还可以是Future对象或者awaitable对象:

1. 如果参数是协程，其实底层还是用的 loop.create_task，返回Task对象
2. 如果是Future对象会直接返回
3. 如果是一个awaitable对象会await这个对象的__await__方法，再执行一次 ensure_future，最后返回Task或者Future

所以就像 ensure_future名字说的，确保这个是一个Future对象：Task是Future 子类，前面说过一般情况下开发者不需要自己创建Future

其实前面说的 asyncio.wait和 asyncio.gather里面都用了asyncio.ensure_future。对于绝大多数场景要并发执行的是协程，所以直接用 asyncio.create_task就足够了~

shield

接着说 asyncio.shield，用它可以屏蔽取消操作。一直到这里，我们还没有见识过Task的取消。看一个例子:

```
In : loop = asyncio.get_event_loop()

In : task1 = loop.create_task(a())

In : task2 = loop.create_task(b())

In : task1.canc
```

```
Out: True

In : await asyncio.gather(task1, task2)

Suspending a

Suspending b

---------------------------------------------------------------------
---------

CancelledError Traceback (most recent call last)

cell_name in async-def-wrapper()

CancelledError:
```

在上面的例子中，task1被取消了后再用 asyncio.gather收集结果，直接抛CancelledError错误了。这里有个细节，gather支持 return_exceptions参数：

```
In : await asyncio.gather(task1, task2, return_exceptions=True)

Out: [concurrent.futures._base.CancelledError(), 'B']
```

可以看到，task2依然会执行完成，但是task1的返回值是一个CancelledError错误，也就是任务被取消了。如果一个创建后就不希望被任何情况取消，可以使用 asyncio.shield保护任务能顺利完成。不过要注意一个陷阱，先看错误的写法：

```
In : task1 = asyncio.shield(a())

In : task2 = loop.create_task(b())

In : task1.cancel()

Out: True
```

```
In : await asyncio.gather(task1, task2, return_exceptions=True)

Suspending a

Suspending b

Resuming b

Out: [concurrent.futures._base.CancelledError(), 'B']
```

可以看到依然是CancelledError错误，且协程a未执行完成，正确的用法是这样的：

```
In : task1 = asyncio.shield(a())

In : task2 = loop.create_task(b())

In : ts = asyncio.gather(task1, task2, return_exceptions=True)

In : task1.cancel()

Out: True

In : await ts

Suspending a

Suspending b

Resuming a

Resuming b

Out: [concurrent.futures._base.CancelledError(), 'B']
```

可以看到虽然结果是一个CancelledError错误，但是看输出能确认协程实际上是执行了的。所以正确步骤是：

1. 先创建 GatheringFuture对象ts
2. 取消任务

3. await ts

asynccontextmanager

如果你了解Python，之前可能听过或者用过contextmanager，一个上下文管理器。通过一个计时的例子就理解它的作用:

```
from contextlib import contextmanager

async def a():

await asyncio.sleep(3)

return 'A'

async def b():

await asyncio.sleep(1)

return 'B'

async def s1():

return await asyncio.gather(a(), b())

@contextmanager

def timed(func):

start = time.perf_counter()

yield asyncio.r
```

```
print(f'Cost: {time.perf_counter() - start}')
```

timed函数用了contextmanager装饰器，把协程的运行结果yield出来，执行结束后还计算了耗时：

```
In : from contextmanager import *

In : with timed(s1) as rv:

...: print(f'Result: {rv}')

...:

Result: ['A', 'B']

Cost: 3.0052654459999992
```

大家先体会一下。在Python 3.7添加了asynccontextmanager，也就是异步版本的contextmanager，适合异步函数的执行，上例可以这么改：

```
@asynccontextmanager

async def async_timed(func):

start = time.perf_counter()

yield await func()

print(f'Cost: {time.perf_counter() - start}')

async def main():

async with async_timed(s1) as rv:

print(f'Result: {rv}')

In : asyncio.ru

Result: ['A', '
```

```
Cost: 3.00414147500004
```

async 版本的with要用 `asyncwith`，另外要注意 `yieldawaitfunc()`这句，相当于yield + `awaitfunc()`

PS: contextmanager和asynccontextmanager最好的理解方法是去看 源码注释，可以看延伸阅读链接2，另外延伸阅读链接3包含的PR中相关的测试代码部分也能帮助你理解

代码目录

本文代码可以在 mp项目 (https://github.com/dongweiming/mp/tree/master/2019-05-24) 找到

延伸阅读

1. https://github.com/python/cpython/blob/3.7/Lib/asyncio/tasks.py#L574
2. https://githu

3. https://github.com/python/cpython/pull/360/

python asyncio理解_深入理解asyncio(二) weixin_29791447的博客 247
Asyncio.gather vs asyncio.wait在上篇文章已经看到多次用asyncio.gather了，还有另...

asyncio之异步上下文管理器 dianyin7770的博客 219
异步上下文管理器 前面文章我们提到了上下文管理器,但是这个上下文管理器只适用...

深入Asyncio（七）异步上下文管理器 weixin_30687811的博客 328
Async Context Managers: async with 在某些场景下（如管理网络资源的连接建立、...

Python asyncio 中await和create_task的区... 最新发布 聆听风雨的博客 963
参考：python - Difference between await Coroutine and await Task - Stack Overflow ...

asyncio 系列一、asyncio 的协程与任务 duxin_csdn的博客 7001
asyncio 的协程与任务 官网：https://docs.python.org/zh-cn/3/library/asyncio-task.h...

asyncio python详解_python协程系列(五)——a... weixin_28917337的博客 240
(1)运行异步协程asyncio.run(coro, *, debug=False) #运行一个一步程序，参见上面(2)...

python协程系列（五）——asyncio的核心概念与基本... MIss-Y的博客 1万+
声明：本文针对的是python3.4以后的版本的，因为从3.4开始才引入asyncio，后面的...

在python里创建一个任务(Task)实例 菜鸟教程 1930
更多编程教程请到：菜鸟教程 https://www.piaodoo.com/ 友情链接： 高州阳光论坛h...

python模块学习 -openpyxl weixin_30897079的博客 188
python模块学习 -openpyxl openpyxl模块介绍 openpyxl模块是一个读写Excel 2010文...

Python 高级编程之 asyncio并发编程 David's Notes 1452
Python 高级编程之 asyncio并发编程1. asyncio 简介1.1 协程与 asyncio1.2 例子 1. as...

async def 函数 的调用 Yq_yang的博客 1万+
import asyncio async def add(x,y): r = x+y return r async def bad_call(a,b,c,d): a_b = a...

python学习随笔(八)_高阶函数_装饰器 weixin_30701575的博客 34
* 待整理高阶函数y = g(f(x))#容器可以比较内容#返回是函数的时候内容比不了,比的...

python编写一个排序函数要求数据输入_python... weixin_39675038的博客 97
1.编写一个程序从文

python装饰器解析

每次调用函数作用域时，您似乎都想向它注入一些变量。一种方法是将数据临时插入...

python异步方法asyncio的使用3 m0_47671192的博客 394
这里我们说下Task 1.task是一个python协程对象，是Future类的一个子类，但它不是...

python里函数定义的理解 大坡3D软件开发 1999
在python里函数定义，与C语言或Java语言的函数定义是不一样的，C语言是定义了...

python学习笔记 咒术高专 59
python笔记1、迭代判断是否可迭代下标迭代生成器2、map/reduce3、filter4、sorte...

python异步装饰器 qq_41661056的博客 573
def calc_time(func): @wraps(func) async def wrapper(*args, **kwargs): start = time....

超常用的Python代码片段 | 备忘单 qq_37289115的博客 326
Python Strings strip lstrip rstrip upper isupper lower islower in not in ord Template St...

async with和async for 热门推荐 flyingzhao 1万+
本文翻译自Python的开发者指南PEP 492。网上async with和async for的中文资料比...

“相关推荐”对你有帮助么？

非常没帮助 没帮助 一般 有帮助 非常有帮助

weixin_39791653
码龄5年 暂无认证

158	-	161万+	20万+	
原创	周排名	总排名	访问	等级

weixin_39791653 关注

0 0 0

私信 关注

搜博主文章

热门文章

java截取字符串的一部分_java中如何截取字符串中的指定一部分 18473

python编程求1!+2!+...+n!_python计算阶乘和的方法(1!+2!+3!+...+n!) 13073

计算机网络设置无法保存,win10系统启用网络发现无法保存的解决步骤 12898

泛微为什么大量招人_800万大学生找不到工作，企业却大喊招人难？背后原因你想不到... 8038

坐标求四面体体积_「体积公式」四面体体积公式 - seo实验室 5221

您愿意向朋友推荐“博客详情页”吗?

强烈不推荐 不推荐 一般般 推荐 强烈推荐

最新文章

容错服务器怎么装系统,企业怎么选择集群服务器和容错服务器

前端预览服务器上文件夹,前端图片预览漫谈

服务器2003系统书,服务器2003系统安装

2021年 154篇 2020年 227篇

weixin_39791653 关注

目录